**Тема:** Використання **Sprites.** Основні команди для роботи з **Sprites.**

**Мета:** Ознайомитись з одним з головних об'єктів гри – спрайтом. Вивчити його функціональні властивості, навчитись керувати спрайтом та динамічно змінювати його параметри.

## Зміст:

1. Основні поняття та створення спрайту.
2. Опорна точка та позиціонування.
3. Ефект обертання
4. Масштабування
5. Кручення
6. Властивості спрайтів, які не залежать від точки-прив'язки спрайту.
   - колір
   - прозорість
   - видимість
7. Поняття полігону.

## Основні поняття та створення спрайту

**Що таке Sprites?**
Sprite - це 2D-зображення, яке можна аніматизувати або трансформувати, змінюючи його властивості, зокрема обертання, положення, масштаб, колір тощо

### Створення Sprites

Існують різні способи створення Sprites залежно від того, що вам потрібно виконати. Ви можете створити Sprite з зображення з різниі графічні формати, включаючи: PNG, JPEG, TIFF та інші.

**Створення спрайту з графічного файлу.**
Sprite можна створити, вказавши файл зображення для використання:

```
auto mySprite = Sprite::create("mysprite.png");
```

мал. 1

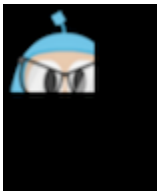
Команда вище створює Sprite, використовуючи зображення mysprite.png. Файл зображення повинен бути розташований в папці Resources. Результат полягає в тому, що спрайтом становиться все що буде знаходитись в файлі, і тиг же оригінальних розмірів. Якщо файл зображення має розміри 200 x 200 пікселів, то створений Sprite становить 200 x 200.

**Створення спрайту з частинки зображення**

У попередньому прикладі створюваний Sprite має той самий розмір, що і оригінальний файл зображення. Якщо ви хочете створити Sprite лише з певної частини зображення, яке міститься в файлі, ви можете зробити це, вказавши відповідну частинку у вигляді прямокутника. Для цього використовується функція **Rect(x,y,w,h).**

Функція **Rect** має 4 значення: початковий **x**, початковий **y**, ширину **w** і висоту **h**:

```
auto mySprite = Sprite::create("mysprite.png", Rect(0, 0, 40, 40));
```



мал. 2

Вирізаний прямокутник починається у верхньому лівому куті. Це трохи незвично, врховуючи те, що система координат в сцені привязана до нижнього лівого кута. Таким чином, результуючий Sprite становить лише частину файлу зображення. У цьому випадку розмір Sprite становить 40 x 40, починаючи з верхнього лівого кута.

Якщо не вказати функцію Rect, Cocos2d-x автоматично використовуватиме повну ширину та висоту вказаного вами файла зображення. Подивіться на приклад нижче. Якщо ми використовуємо зображення розміром 200 x 200, наступні два записи матимуть однаковий результат.

```
auto mySprite = Sprite::create("mysprite.png");
auto mySprite = Sprite::create("mysprite.png", Rect(0, 0, 200, 200));
```

**Створення спрайтів з спрайт-листів.**

Ви хочете використовувати різні спрайти та анімації у вашій грі. Найтиповіший підхід - додати їх до роекту і створити цикл, який завантажує їх один за одним.

Це погана ідея з кількох причин:

- завантаження кожного файлу вимагає часу

- кожен спрайт повинен бути завантажений у графічну пам'ять окремо
- перемикання між текстурами забирає процесорний час
- не можна оптимізувати спрайт, щоб зменшити перевитрати

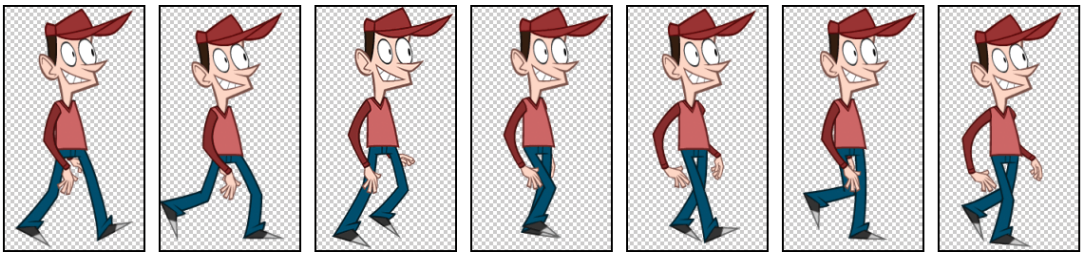
Спрайт-лист - це спосіб об'єднання спрайтів в єдиний файл. Використання спрайт-листів допомагає досягти кращої продуктивності шляхом групування однотипних спрайтів в один файл. Вони також можуть економити дискову та відеопам'ять у випадках, коли спрайт можна запакувати в спрайт-лист більш ефективно (зазвичай потрібні спеціальні інструменти).

При використанні спрайт-листів вони перш за все завантажуються в SpriteFrameCache. SpriteFrameCache - клас кешування, який зберігає об'єкти SpriteFrame, додані до нього, для майбутнього швидшого доступу. SpriteFrame завантажується один раз і зберігається в SpriteFrameCache.

Ось кілька прикладів типових спрайт-листів:

а)

б)



в)

мал 3 (а,б,в)

Створювати спрайт-листи можна використовуючи TexturePacker

***Важливо.***

*Спрайт-листи створюються з окремих графічних файлів кожен з яких, очевидно має свою назву. Саме по цих назвах відбуватиметься ідентифікація окремих спрайтів з спрайт-листа при їх використанні в проекті гри..*

Як ви бачите спрайт лист, як мінімум об'єднує всі спрайти в єдиний файл.

**Завантаження Sprite-листа.**

Завантажте ваш спрайт лист у SpriteFrameCache, ймовірно, в AppDelegate:

```
// створення спрайт-листа
auto spritecache = SpriteFrameCache::getInstance();

// підвязування спрайт-листа
spritecache->addSpriteFramesWithFile("sprites.plist")
```

Завантаження спрайт-листа повинне відбуватись перед їхнім використанням. Як правило це варто зробити в файлі ***.cpp в функції ініціалізації: `bool HelloWorld::init()` перед `return true;`

Тепер, коли у нас є спрайт-лист, завантажений в SpriteFrameCache, ми можемо створити об'єкти Sprite, використовуючи його.

```
auto mysprite = Sprite::createWithSpriteFrameName("mysprite.png");
```

**Створення спрайту із фрейму.**

Інший спосіб створити той самий Sprite - залучення SpriteFrame з SpriteFrameCache, а потім створення Sprite за допомогою SpriteFrame. Приклад:

```
auto newspriteFrame = SpriteFrameCache::getInstance()-
>getSpriteFrameByName("Blue_Front1.png");
auto newSprite = Sprite::createWithSpriteFrame(newspriteFrame);
```

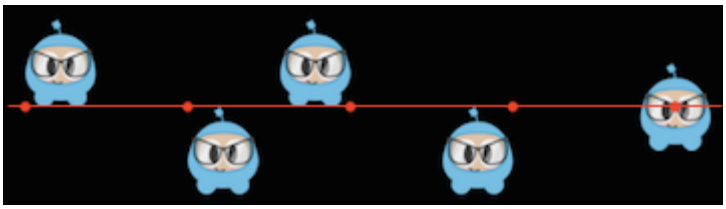
мал. 4

# Опорна точка та позиціонування

Опорна точка - це точка, яку ви встановили як спосіб вказати, яку частину спрайта використовуватимете при встановленні його положення. Точка прив'язки впливає тільки на ті властивості, які можна перетворити. Це включає масштаб, обертання, скручення. Це невпливаєна колір та прозорість. У опорній точці використовується нижня ліва система координат. Це означає, що, коли ви задаєте значення координат X і Y, вам слід обов'язково починати з нижнього лівого кута, щоб зробити ваші розрахунки. За замовчуванням всі об'єкти вузла мають за замовчуванням опорну точки (0.5, 0.5). Тобто середину спрайту.
Встановити опорну точки легко:

```
// опорна точка по замовчуванню
mySprite->setAnchorPoint(0.5, 0.5);

// нижня ліва
mySprite->setAnchorPoint(0, 0);

// верхня ліва
mySprite->setAnchorPoint(0, 1);

// нижня права
mySprite->setAnchorPoint(1, 0);

// верхня права
mySprite->setAnchorPoint(1, 1);
```

Візуалізація вищенаписаного коду відображена на мал. 5:



мал. 5

## Position

Позиція спрайта впливає на її опорну точку, оскільки саме ця точка використовується як відправна точка позиціонування. Давайте візуально подивимося, як це відбувається. Зверніть увагу на кольорову лінію і місце, де знаходиться спрайт. Зверніть увагу: коли ми змінюємо значення опорної точки, позиція спрайта змінюється. Важливо зазначити, що все, що потрібно, змінює значення опорної точки. Ми не використовували команду setPosition() для досягнення цього мал. 5.

Існує більше способів встановити позицію, ніж просто опорну точку. Sprite-об'єкти також можна встановити, використовуючи метод setPosition ().

```
// розміщення спрайта в позицію x = 100, y = 200.
mySprite->setPosition(Vec2(100, 200));
```

## Розміри спрайту

Щоб отримати позицію спрайта, використовується наступний код:

```
Vec2 point = sprite->getPosition(); float x = point.x; float y = point.y;
```

Щоб отримати розмір спрайта, використовується наступний код:

```
Size size = sprite->getContentSize(); float width = size.width; float height = size.height;
```

За замовчуванням спрайт позиція (0, 0). Ви можете змінити позицію спрайту за допомогою методу setPosition та отримати його за допомогою методу getPosition. Ви можете отримати розмір спрайта, використовуючи метод getcontentization. Тим не менше, змінити розмір спрайта не можна за допомогою методу setContentization. Contentsize - це константне значення. Якщо є необхідність щмінювати розмір спрайта, то слід змінювати масштаб спрайту. Правда така процедура має свій недолік. При маштабувані може втрачатись чіткість та якість зображення.

## Прямокутник

Для отримання спрайт-прямокутника, тобто згенерувати прямокутник по розміру спрайта, слід використовувати наступний код:

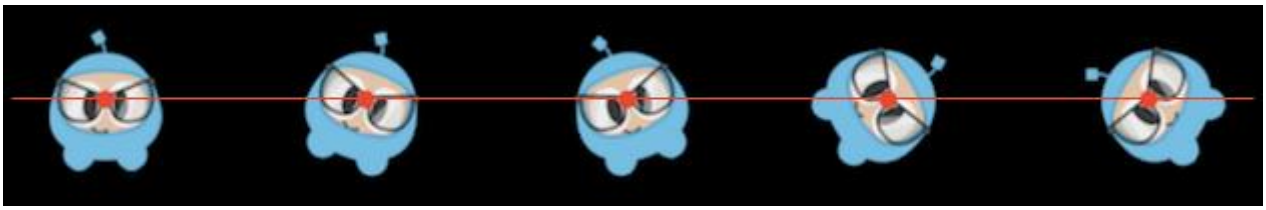
```
Rect rect = sprite->getBoundingBox();
Size size = rect.size;
Vec2 point = rect.origin;
```

Rect - спрайт прямокутник, який має такі властивості, як розмір і Vec2. Якщо шкала не дорівнює одному, то розмір у Rect не буде рівним оригінальному розміру, використовуючи метод getContentization, розмір getContentize є вихідним розміром зображення. З іншого боку, розмір в Rect за допомогою getBoundingBox - це розмір зовнішнього вигляду. Наприклад, коли ви встановлюєте спрайт на половину маштабу, розмір в Rect, використовуючи getBoundingBox, становить половину розміру, а розмір, який використовує getContentsize, є оригінальним розміром. Позиція та розмір спрайта є дуже важливим моментом, коли потрібно розмістити спрайт на екрані.

# Ефект обертання.

Метод `setRotation` здійснює обертання спрайта за годинниковою стрілкою чи проти годинникової стрілки, в залежності від додатнього чи від'ємного значення аргументу. Додатній аргумент обертає об'єкт Sprite за годинниковою стрілкою, а від'ємне значенн аргумента призводить до обертання об'єкту Sprite проти годинникової стрілки. Значення за замовчуванням дорівнює 0.

```
// поворот спрайту на 20 градусів за годинниковою стрілкою
mySprite->setRotation(20.0f);

// поворот спрайту на 20 градусів проти годинникової стрілки
mySprite->setRotation(-20.0f);

// поворот спрайту на 60 градусів за годинниковою стрілкою
mySprite->setRotation(60.0f);

// поворот спрайту на 60 градусів проти годинникової стрілки
mySprite->setRotation(-60.0f);
```

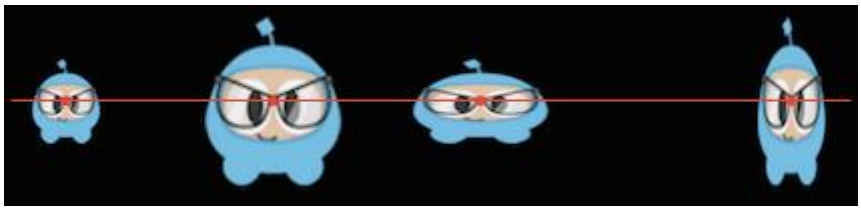


мал. 6

## Масштабування

Змінює масштаб спрайту по осі x(ширина), чи по осі y(висота) або рівномірно для x і y. Значення за замовчуванням дорівнює 1,0 для x і y.

```
// збільшення розміру спрайту в 2 рази
mySprite->setScale(2.0);

// зміна ширини спрайту в 2 рази
mySprite->setScaleX(2.0);

// зміна висоти спрайту в 2 рази
mySprite->setScaleY(2.0);
```

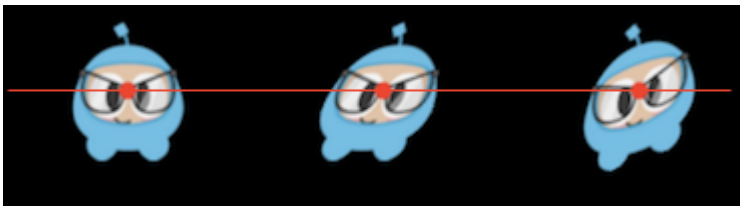


мал. 7

## Кручення

Видозмінення спрайту, використовуючи ефект кречення, аналогічно до маштабування, зміна може відбуватись як по висоті так і по ширині, або одночасно по висоті та ширині. Значення за замовчуванням - 0,0 для x і y.

```
// корректування по ширині на 20
mySprite->setSkewX(20.0f);

// корректуваня по висоті на 20
mySprite->setSkewY(20.0f);
```



мал. 8

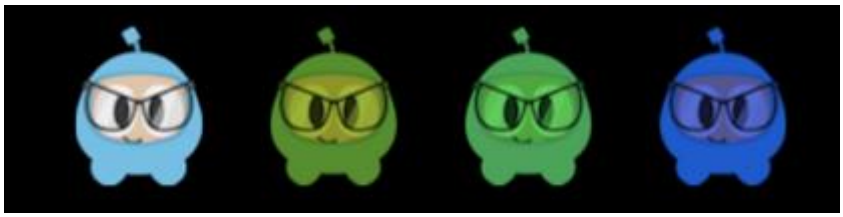
## Властивості спрайтів, які не залежать від точки-прив'язки спрайту.

Всі вищеописані властивості зміни спрайтів залежать від точки прив'язки спрайта. Такі вдластивості спрайтів, як колір та яскравість не залежать від точки привязки.

### Колір

Зміна кольору спіра. Це робиться шляхом використання функції Color3B застосовуючи її до об'єкта спрайту. Функція Color3B – має значення RGB. Color3B це просто функція, яка визначає колір RGB. Колір RGB має значення 3 байтів від 0 до 255. Cocos2d-x також забезпечує попередньо визначені назви кольорів, які можна вибирати. Використання цих параметрів буде трохи швидше, оскільки вони заздалегідь визначені. Кілька прикладів: Color3B :: White та Color3B :: Red.

```
// встановлення кольору використовуючи наперед визначену назву.
mySprite->setColor(Color3B::WHITE);
// встановлення кольору використовуючи RGB код кольору.
mySprite->setColor(Color3B(255, 255, 255)); // аналог Color3B::WHITE
```



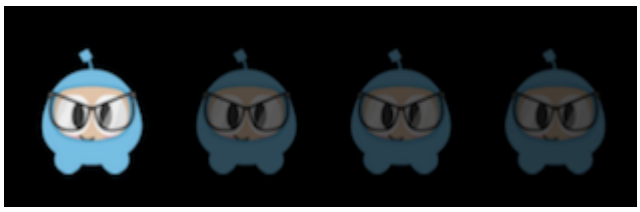
мал. 9

### Прозорість

Змінює рівень прозорості спрайта на вказане значення. Об'єкт по замовчуванню не є прозорим взагалі. Ця властивість має значення від 0 до 255, де 255 означає, що він повністю непрозорий, а 0 - повністю прозорий. Подумайте:

нульове значення означає невидимість. Значення за замовчуванням 255 (повністю непрозорі).

```
//Встановлення прозорості 30, таке значення робить спрайт на 11.7% непрозорим.
// (30 ділимо на 256 отримаємо 0.1171875...)
mySprite->setOpacity(30);
```



мал. 10

### Видимість

Можна змінити видимість спрайту, передаючи логічне значення. Якщо аргумент хибний (false), то спрайт невидимий; якщо аргумент істинний (true), тоді спрайт видно. Значення за замовчуванням завжди відповідає істиності.

```
sprite->setVisible(false);
```

Якщо потрібно перевірити видимість спрайту, то використовується метод isVisible, а не методом getvisible. Клас спрайту не має методу getvisible.

```
if (sprite->isvisible()) {
        // visible
}
else {
        // invisible
}
```

## Polygon Sprite

Починаючи з версії cocos2d-x 3.9 існує нова функціональна можливість представлення спрайтів: **Polygon Sprite**. Замість того, щоб малювати цілий багатокутник, фарбуються лише непрозорі частини, що збільшує частоту кадрів вашої гри.

Для ефективного використання полігону, існує необхідне програмне забезпечення, яке створює такі спрайти, наприклад TexturePacker 4.0.0.

Використання полігонів дозволить:

- підвищення продуктивності - використання багатогранних спрайтів;
- зменшення використання пам'яті - використання стиснутих папок спрайт листів.

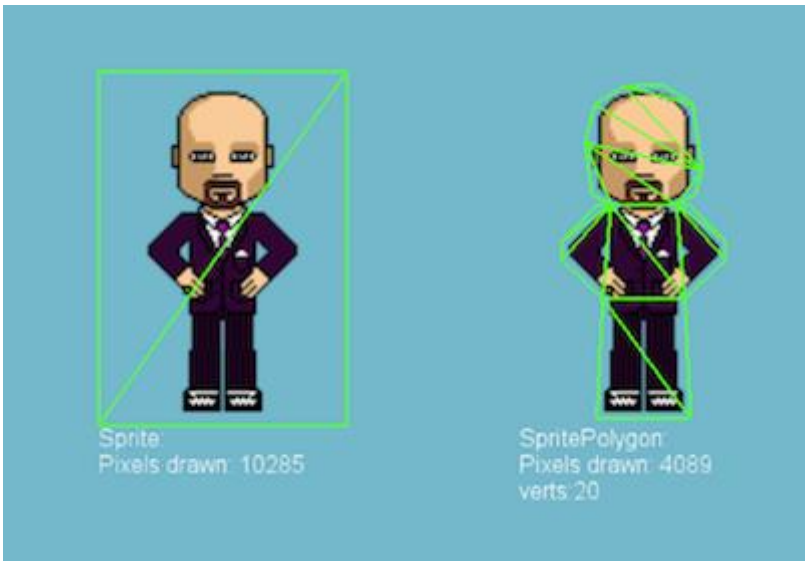
**Polygon Sprite** також є спрайтом, який використовується для відображення 2D зображення. Однак, на відміну від звичайного Spriteobject, прямокутника,

зробленого лише з 2-х трикутників, об'єкти PolygonSprite виконані із серії трикутників.

Навіщо користуватись багатокутним спрайтом?

Відповідь проста - це продуктивність!

Існує багато технічних прийомів, які ми можемо розібрати тут щодо швидкості заповнення пікселів, але PolygonSprite малює на основі форми нашого Sprite сукупність трикутників, а не простого прямокутника навколо найбільшої ширини та висоти. Це заощаджує багато непотрібного місця. Розглянемо цей приклад:




мал. 11

Зверніть увагу на різницю між лівими та правими версіями!

Зліва, типовий Sprite, намальований прямокутним способом за допомогою 2 трикутників.

Праворуч, PolygonSprite, намальований багатьма дрібними трикутниками.

Незалежно від того, чи доцільний такий компроміс з чисто економічного боку де вигода залежить ще від ряду факторів (форма спрайту, деталізація, розмір, кількість на екрані тощо), але в цілому вершини продуктивніші, ніж пікселі на сучасних графічних процесорах.

## Аутополігон

AutoPolygon є допоміжним класом. Мета полягає в тому, щоб обробляти зображення в 2d полігоні сітки під час виконання.

Є функції для кожного етапу процесу, відстеження всіх точок, до триангуляції. Результат може бути переданий об'єктам Sprite, створюючи функцію, щоб створити PolygonSprite.

```
// Generate polygon info automatically.
auto pinfo = AutoPolygon::generatePolygon("filename.png");

// Create a sprite with polygon info.
auto sprite = Sprite::create(pinfo);
```

# Список літературних джерел по темі:

1. **Siddharth Shekar** Learning Cocos2d-x Game Development // 2014 Packt Publishing, p.266, ISBN 978-1-78398-826-6
2. Raydelto Hernandez Building Android Games with Cocos2d-x // 2015 Packt Publishing, p. 160, ISBN 978-1-78528-383-3
3. Roger Engelbert Cocos2d-x by Example Beginner's Guide Second Edition // 2015 Packt Publishing, p. 270, ISBN 978-1-78528-885-2
4. Akihiro Matsuura Cocos2d-x Cookbook // 2015 Packt Publishing, p. 255, ISBN 978-1-78328-475-7
5. Karan Sequeira Cocos2d-x Game Development Blueprints // 2015 Packt Publishing, p. 392, ISBN 978-1-78398-526-5
6. Frahaan Hussain, Arutosh Gurung, Gareth Jones Cocos2d-x Game Development Essentials // 2014 Packt Publishing, p. 392, ISBN 978-1-78398-786-3